Introduction

PCA

1. Normalize input data
2. Compute k orthonormal vectors (i.e. principal components), which will be sorted in order of decreasing significance
3. Rewrite each input datapoint as a linear combination of the k principal components

t-SNE

1. Define a probability distribution over pairs of high-dim data points: similar data points have a high probability of being picked, dissimilar data points have small probability
2. Define a similar distribution over the points in the map space
3. Minimize the Kullback-Leibler divergence between the two distributions w.r.t. the locations of the map points (through gradient descent)

Association Rules

Apriori

1. Set the minimum support (minsup) and k=1
2. Generate all possible k-dim itemsets: if k>1 consider only frequent itemsets from step k-1
3. Compute the support for all the k-dim itemsets
4. Prune the itemsets with support < minsup
5. If there are no new frequent itemset the procedure ends, otherwise k++ and go to 2.

Eclat algorithm: (↑) but with the vertical databse

FP-tree

1. Sort the single items based on their support
2. Reorder each transaction based on 1.
3. Add each transaction one by one into the FP-tree and keep track of the count at each node
4. For each item i

   → compute the conditional FP-tree: seach all the paths that end with i, build the conditional FP-tree considering the count of i in the paths

   → based on the conditional FP-tree of item i and considering the minimum support, generate all the frequent itemsets containing i

Trawling: searching for small communities

1. Graph to database: for every node we create an itemset as all the nodes the node is pointing
2. Database to frequent itemsets (any method)
3. Frequent itemsets to subgraphs (fully connected bipartite graphs): for any frequent itemsets put the nodes of the itemset on the right and put on the left only the nodes which are connected to *all* the itemset-nodes

GSP algorithm: as the apriori algorithm, but for sequences (not itemsets), we consider the order

Clustering

Hierarchical clustering

1. Compute proximity matrix of pairwise distances between all the points
2. Let each data point be a cluster
3. Merge the two closest clusters
4. Update proximity matrix
5. If there is a single cluster the procedure ends, otherwise go to 3.

k-means

1. (Given k) Randomly initialize k centroids
2. Assign each point to the closest centroid
3. Update centroids (as mean of their *new* clusters)
4. If stopping conditions are met the procedure ends, otherwise go back to 2.

Mean shift clustering

1. Choose a search window size (bandwidth)
2. For each point of the dataset:

   a. center the search window at the point

   b. compute the mean of the search window

   c. center the search window at the mean (*b.*)

   d. if stoppings conditions are met (the shift was small) go to the next point, otherwise go to *b.*
3. Clustering step: assign points that lead to nearby modes to the same cluster

Expectation maximization (EM) clustering

1. Initialize the estimate of the parameter vectors
2. For each cluster, use the current estimate of the parameters to compute the posterior probabilities: $\mathbb{P}(C_i|\vec{x}_j) = \mathbb{P}(\vec{x}_j \in C_i)$
3. Update for all i: $\vec{\mu}_i, \Sigma_i, \mathbb{P}(C_i)$
4. If stopping conditions are met the procedure ends, otherwise go back to 2.

DBSCAN

The algorithm needs to compute the ϵ-neighborhood for each point. Once it has the neighborhoods, the algorithm needs only a single pass over all the points to find the denisty-based clusters.

HDBSCAN

1. Transform the space

The distance between x, y in the new space will be: $d_{new}(x, y) = \max\{core_k(x), core_k(y), d(x, y)\}$, where $d(\cdot, \cdot)$ is the original distance metrics and the core distance of x is defined as the maximum distance of a x to its k-th nearest neighbor.

2. Build the minimum spannin tree

Build the minimum spanning tree: data points are vertices, an edge between x, y has weight equal to $d_{new}(x, y)$. Build the tree one edge at the time by adding the lowest weight edge that connects the current tree to a point not yet in the tree.

3. Build the cluster hierarchy

Convert the minimum spanning tree to a hierarchy of connected components. Sort the edges of the tree

by weight (distance) in increasing order, and iterate creating a new merged cluster for each edge.

4. Extracting clusters

Condense the dendrogram to highlight the difference between mergings. Procede iteratively starting from the biggest cluster. At each split, check the size of the two new clusters: if a cluster has fewer points than a treshold then it's eliminated, otherwise both clusters are maintained. Given the condensed dendrogram, select clusters that last long.

Classification

Logistic regression

Define a score function:

$$Score(\vec{x}_i) = \sum_{j=1}^{D} w_j \, h_j(\vec{x}_j)$$

where $h_j(\vec{x}_i)$ is the preprocessed input.

Compute the $\mathbb{P}$ of assigning a class to the point:

$$\mathbb{P}(\hat{y}_i = +1|\vec{x}_i) = \frac{1}{1+e^{-Score(\vec{x}_i)}} = \frac{1}{1+e^{-\vec{w}\,\vec{h}(\vec{x}_i)}}$$

To optimize, logistic regression searches for weights that correspond to the highest likelihood (it aims to maximize the product of all the probabilities):

$$l(\vec{w}) = \prod_{i=1}^{N} \mathbb{P}(y_i|\vec{x}_i, \vec{w}_i)$$

For this purpose, it is convenient to use the log likelihood and to apply the gradient ascent.

To classify a point we select the class with the highest probability. Equivalently, we can check the ratio of the probabilities and assign $+1$ if:

$$\frac{\mathbb{P}(\hat{y}_i=+1|\vec{x}_i)}{\mathbb{P}(\hat{y}_i=-1|\vec{x}_i)} = e^{\vec{w}\,\vec{h}(\vec{x}_i)} > 1$$

$$\log\left(\frac{\mathbb{P}(\hat{y}_i=+1|\vec{x}_i)}{\mathbb{P}(\hat{y}_i=-1|\vec{x}_i)}\right) = \vec{w}\,\vec{h}(\vec{x}_i) > 0$$

Models comparison

1. Generate k folds for each model: $\theta_1^A, .., \theta_k^A, \theta_1^B, .., \theta_k^B$
2. Compute differences, mean and std deviation:
   $\delta_i = \theta_i^A - \theta_i^B$, $\mu_\delta = \frac{1}{k}\sum_i \delta_i$, $\sigma_\delta = \sqrt{\frac{1}{k}\sum_i (\delta_i - \mu_\delta)^2}$
3. Set a confidence level α and test:
   $H_0 : \mu_\delta = 0$ vs. $H_1 : \mu_\delta \neq 0$
   if p-value $> \alpha$ the difference in performance is not statistically significant (we do not reject H_0)

Naive Bayes classifiers

Given $\vec{x}$, look for the class with highest probability:

$$class = \arg\max_y \mathbb{P}(y|\vec{x})$$

Naive Bayes approach:

$$\mathbb{P}(y|\vec{x}) = \frac{\mathbb{P}(x_1|y)\,..\,\mathbb{P}(x_n|y)\,\mathbb{P}(y)}{\mathbb{P}(\vec{x})}$$

To perform the ***training*** count the frequency of touples (x_i, y) for each attribute value x_i and each class value y in the dataset. Then use counts to compute estimates for the class probability $\mathbb{P}(y)$ and the conditional probability $\mathbb{P}(x_i|y)$. To ***test***, given a new point $\vec{x}$ compute the most likely class as:

$$\begin{aligned} class &= \arg\max_y \mathbb{P}(y|\vec{x}) \\ &= \arg\max_y \mathbb{P}(x_1|y)\,..\,\mathbb{P}(x_n|y)\,\mathbb{P}(y) \end{aligned}$$

k-Nearest neighbors classifier

1. Compute distance to other training records
2. Identify the k nearest neighbors
3. Use labels of nearest neighbors to determine the label of the point (majority voting/others)

Decision tree

1. Initially all training instances are at the root
2. Go to a leaf: if it stopping conditions are met, go to another leaf. If all leafs satisfy stopping conditions, the procedure ends.
3. Given a leaf, evaluate all the available attributes through information gain
4. Split the group of points based on the best attribute and, if categorical, remove the attribute from the available attributes of the children

Ensemble Methods

Bagging (bootrsap aggregation)

1. Consider a dataset of D touples
2. At iteration i sample with raplcement from D a training set D_i of d touples (bootstrap)
3. Learn a model M_i for each training set D_i
4. Return the target prediction for each model M_i
5. The bagged model M^* returns as final result the majority in case of classification or the average in case of regression

Random forests

1. Consider a dataset of D touples
2. At iteration i sample with replacement from D a training set D_i of d touples (bootstrap)
3. Learn tree T_i from D_i using at each node only a subset of the n variables, without pruning
4. Return the target prediction for each tree T_i
5. The output is computed as the majority voting for classification or average for prediction

Boosting – *focus on missclassified points*
Adaboost – classification with two classes $(-1/+1)$.

1. Assign uniform weights to each training sample
2. At iteration i learn a weak classifier h_i
3. Compute the error $\epsilon_i = \sum_j w_j \mathbb{I}_{\{h_i(x_j) \neq y_j\}}$
4. Compute $\alpha_i = \frac{1}{2} \ln(\frac{1-\epsilon_i}{\epsilon_i})$
5. Update the weights:
$$w_{i+1} = w_i e^{-\alpha_i} \quad \text{correctly classified}$$
$$= w_i e^{\alpha_i} \quad \text{incorrectly classified}$$
and normalize them
6. The final model is:
$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x)) \ \in \{-1, +1\}$$

Gradient boosting – *focus on target and residuals*

1. Learn a basic (even simple) predictor
2. Compute the gradient of a loss function w.r.t. the predictor, for instance the mean square error
$$MSE(y, \hat{y}) = \frac{1}{N} \sum_i (y_i - \hat{y})^2$$
3. Compute a model to predict the residuals
4. Update the predictor with a new model
$$\hat{y}_i = \hat{y}_i + \alpha \nabla MSE(y, \hat{y})$$
where large α means larger steps
5. If stopping conditions are met the procedure ends, otherwise go to 2.

Genetic Algorithms

1. Initial population
2. Evaluation (given a fitness function)
3. Tournament selection: select a random subset of k solutions from the original population and then select the best solution out of the subset
4. Variation (given two solutions)
   → One-point crossover
   → Two-points crossover
   → Uniform crossover
   → Bit-flip mutation
5. Replace: replace all, replace worst (elitism)